

## Instructions

Complete the problems assigned below and turn in your answers by the due time above. To receive full credit, you must show both the relevant command(s) and output. If a question asks about what a command does, or what the output would be, give both the command and the output. Demonstrate that you actually performed the command, and didn't just guess.

**Requirements** - All of your shell programs henceforth must:

- use the shell template
- check for valid arguments
- output sensible *usage* or syntax messages
- be appropriately commented
- exit with an appropriate exit status

### Homework Problems

- Modify your rename program from last week to use the **getopts** command instead of parsing the arguments manually.

Here is a contextual **diff** showing the difference between the original version without using getopts and the new version using getopts. It would be good to become familiar with the output of diff, as it is used frequently. Lines preceded with ! are changed lines – the top hunk is the original code and the bottom hunk is the new code. The changed line number range is shown above each hunk of code. In other words, lines 27 through 52 in the original version of the code code (top) were replaced by lines 27 through 48 in the new code (bottom). The lines preceded with a – in the original code are lines that were removed in the new code and lines preceded with a + in the new code are new lines.

```
$ diff -c ../hw6/rename rename
*** ../hw6/rename      Tue Feb 19 17:56:16 2002
--- rename             Sat Feb 23 16:08:14 2002
*****
*** 27,52 ****
!
!   # First, process the options.  Just remember what was found by use of
!   # variables - this will indicate what translations to perform later.
! while [ $# -ne 0 ] ; do
!     case $1 in
!       -u) filecase=upper ;;
!       -l) filecase=lower ;;
!       -w) removewhite=1 ;;
!       -v) verbose=1 ;;
!       -*)
!         # It is common practice that anything starting with a dash that
!         # isn't one of the allowable options is an error.
!         echo Unknown option \"$1\"
!         echo $usage
!         exit 1
!       ;;
!     *)
!       break          # presume filelist has started
!     ;;
!   esac
-   shift              # option processed, now shift it out
done

!
!   # Now that the options are out of the way, we can process the remaining
!   # arguments, which are assumed to be the list of files.
!   # The strategy here is to first calculate the new name of the file, and
--- 27,48 ----

!
!   # First, process the options.  Just remember what was found by use of
!   # variables - this will indicate what translations to perform later.
! while getopts ulvw option ; do
```

```

!     case "$option" in
!       u) filecase=upper ;;
!       l) filecase=lower ;;
!       w) removewhite=1 ;;
!       v) verbose=1 ;;
!       \?)
!         echo $usage
!         exit 1
!       ;;
!     esac
done

+ # shift away the options just processed
+ shift `expr $OPTIND - 1`
+
# Now that the options are out of the way, we can process the remaining
# arguments, which are assumed to be the list of files.
# The strategy here is to first calculate the new name of the file, and

```

2. Modify your trash program (trm, etc.) to handle the following additional requirements:

- a. Correctly handle path names. Currently, if the arguments contain paths (i.e. ../foo or \$HOME/hw6/dir1) you will still place your files in the trash in the current working directory. But files should be placed in the local trash in the directory where they live, not in the current working directory (i.e. ../foo should be placed in ../.trash/foo and \$HOME/hw6/dir1 should be placed in \$HOME/hw6/.trash/dir1). Thus, all trashed files will remain within the same directory tree. This will allow you to restore files to their correct locations as well.
- b. Implement a **-i** option to interactively ask for confirmation before overwriting any files.
- c. Implement a **-f** option to force overwriting any files without question – this overrides **-i** above.
- d. Use **getopts** to process options

Here is one solution that satisfies the requirements:

```

$ cat trm
#!/bin/sh

# Implements trash can functionality, that places removed items
# in a local trash for possible future recovery.

# Three functions are implemented:
#   trm
#     removes items specified on the command line, by placing them
#     into the trash in the local directory
#   trecov
#     recovers the command line specified items from the trash in
#     the item's parent directory
#   tempty
#     empties and removes the local trash can
#
# Error codes: 0 = success; 1 = Usage, 2 = error

trash=.trash          # name of the directory-local trash can

prog=`basename $0`

# tempty
if [ $prog = "tempty" ] ; then
  if [ $# -ne 0 ] ; then
    echo $prog: too many arguments
    echo usage: $prog
    exit 1
  fi

  # no need to complain if there is no trash; the job is done either way
  rm -rf $trash
  exit 0

```

```

fi

# trm and trecov
#
usage="usage: $prog [-fi] path [...]"

# Values for overwrite variable:
#     NULL : existing file produces an error (default)
#     -f   : forces overwrite without question
#     -i   : ask for confirmation before overwrite
overwrite=

# Process the options, assigning the overwrite variable with
# the option that will be passed to mv later. Letting the options
# accumulate allows mv's default -f overrides -i behavior.
while getopts if option ; do
    case "$option" in
        f) overwrite="$overwrite -f" ;;
        i) overwrite="$overwrite -i" ;;
        \?) echo $usage ; exit 1 ;;
    esac
done
shift `expr $OPTIND - 1`      # shift away the options just processed

if [ $# -eq 0 ] ; then
    echo $prog: too few arguments
    echo $usage
    exit 1
fi

# assume good exit status, until proven otherwise
status=0

for i in "$@" ; do
    trashdir=`dirname $i`/$trash
    trashedfile="$trashdir/`basename $i`"

    # trm and trecv are perform essentially the same job - that is,
    # moving files from on directory to the other. Appropriately assigning
    # the variables "to" and "from" allows use of a generalized mv command
    if [ $prog = "trm" ] ; then
        to="$trashedfile"
        from="$i"

        # make the directory-local trash as needed
        mkdir -p "$trashdir"
        tmpstatus=$?
        if [ $tmpstatus -ne 0 ] ; then
            echo $prog: Error: $status : failed to make trash $trashdir
            status=tmpstatus
            continue;
        fi
    else
        to="$i"
        from="$trashedfile"
    fi

    if /bin/test ! -e "$from" ; then
        echo $prog: $from does not exist
        status=2
        continue
    fi

    # if overwrite mode is unset (default), then we can't overwrite
    # existing files, so this should produce an error. Because
    # the mv command overwrites by default when in a shell program
    # (i.e. STDIN is not a terminal), we must process this ourselves
    if /bin/test -z "$overwrite" -a -e "$to" ; then

```

```
        echo $prog: $to exists! To overwrite, use -f or -i
        status=2
        continue
    fi

    /usr/bin/mv $overwrite "$from" "$to"
    tmpstatus=$?
    [ $status -eq 0 ] && status=$tmpstatus

    if [ $prog = "treco" ] ; then
        # trying to remove a non-empty trash will fail, but this is desired!
        rmdir $trashdir 2> /dev/null
    fi
done

exit $status
```